COMP 520: Compilers
Compiler Project – Final submission

Due: Wed Apr 27, 11:59 PM

The final submission of the project adds no new functionality, but provides an opportunity to correct PA1-PA4 errors in your compiler and make improvements as you wish. It also provides an opportunity to add extensions to your compiler for extra credit. *The extra credit is not needed - all grades can be obtained with just the basic project* (an exception is a team project which earns score at a lower rate). The following details the parts of your submission that you should place in your `final` submission directory

1. Guide to your compiler

This is a short document titled `guide.pdf` or `guide.txt` that you place in your final submission directory. The document should contain the following:

- **Scope of your project**. Please make clear any optional parts of the project you have implemented, if any. List known limitations of your implementation (it is better for you to identify these than for me to find them).
- **Summary of changes to AST classes**. Summarize any changes you made to the `AbstractSyntaxTree` classes that were distributed. Describe changes, if any, made to the AST class structure as well as additions made to the classes to support contextual analysis and code generation.
- **A description of tests (if tests are supplied)**. This should describe any tests in the Tests directory (see below). If you are not able to run `PA4Test.java` you should include some test(s) that exercise the portion of the compiler that is working. If you have completed optional extensions of the project, you should include some comprehensive tests for each extension. You can follow the `passxxx`, `failxxx` convention we have been using for tests that should demonstrate correct behavior from your compiler and those that should be rejected by your compiler.

2. miniJava Compiler

Place a copy of your `miniJava` directory in the submission directory. Your compiler should compile miniJava programs supplied as files with extension ".java". For example, if `test35.java` is a valid miniJava program, your compiler should terminate with `exit(0)` and generate object code file `test35.mJAM`.

If the source program *is not* a valid miniJava program, your compiler should write a diagnostic message and `exit(4)`. Error messages beyond syntactic analysis need to be prefixed with "***".

The operation of your compiler should be as specified in PA1-PA4 except where it conflicts with any extensions added.

3. Tests

This directory (if present) contains any special test programs described in your guide.

Grading

The base functionality of the final project, as specified in PA4, will be assessed by functional testing, inspection of the generated code, and inspection of generated diagnostics. The overall score of the project is obtained by combining contributions from all five checkpoints to obtain a normalized project score out of 100.

If you wish, you may earn additional points to add to the overall score by incorporating further features of Java into miniJava, as shown below. Unless indicated otherwise, the intended semantics of each feature correspond to Java semantics, and, in general, may involve extensions or changes in all phases of your compiler.

You can get a fine grade without attempting any extension, so it is perfectly reasonable (and the typical choice) to concentrate on finalizing the base project. If you do want to investigate an extension, make sure to think it through in advance by checking Java semantics. Some extensions may be more work than their point value suggests.

Point value	Feature
2	Static field initialization.
2	Parameterized class constructors. There should only be one constructor per class, but it may have parameters. If none is defined, the default constructor should be available.
3	**for** loops. Be sure to consider the possible forms of the initialization (including declaration of the iterator variable), loop test, and increment parts.
1-3	Improve code generation for the condition (test) in **while** and **if** statements, focusing on efficient evaluation of short-circuit boolean operators **&&** and **||**. Ideally, efficient evaluation means: (1) minimize alternation between jumps and construction of truth values on the stack and (2) no chains of consecutive jumps without intervening tests in the evaluation of a conditional expression. Partial credit is available for anything that improves on the base strategy.
4	Add the String type and string literals. No operations need to be supported on strings, but you must be able to assign a string literal or a String reference to a variable of type String, and it must be possible to print String values by overloading `System.out.println()`.
5	Add overloaded methods that differ in the types of their arguments, and perform type checking to determine their validity and to resolve overloading.
10–20	Inheritance of fields and methods, and dynamic method invocation. Be sure type checking is extended appropriately. Optionally support `instanceof` and/or `super()`.